Response to Call for SET Proposals

Automated Test Case Generator

Yu-Wen Tung
Section 368
818-354-2673
Yu-Wen.Tung@jpl.nasa.gov
Supervisor: E. Jay Wyatt

1. Project Summary

The objective of this proposal is to develop a parametric test case generator, called TCG, to be used by MDS, X2000, OPSP and other JPL mission projects. This tool is meant to cut down the number of tests needed, sometimes dramatically, yet it would guarantee a complete coverage. The prototype of this tool has shown a very promising performance comparable to the best commercial tool AETG, which confirms that the TCG algorithm is on a par with the state of the art. By implementing this algorithm as a reusable software component it can then be embedded in a variety of test harnesses and can work synergistically with other test tools (while a tool like AETG cannot). This helps a project's verification effort apply a consistent, disciplined approach to achieving a selected degree of test coverage with a near-minimal number of test cases.

The proposed work will not only transform the TCG prototype into a robust software product, it will also extend the algorithm and the tool with many important features. These features are valuable as customers enter a more comprehensive phase of testing. "Seed cases" enable engineers to specify mandatory test spaces, ranging from partially specified test combinations to completely specified test cases, thus ensuring that certain combinations *always* get tested. N-way coverage enables engineers to increase the degree of coverage when testing particularly critical elements. Constraints enable engineers to eliminate combinations that represent physical or logical impossibilities. We believe the resulting TCG tool will become an important contributor to effective and automatic testing in the coming generation of mission software systems.

The cost of implementing this reusable tool with desired features is estimated as six man-months. This low cost is possible because both the algorithm and the software architecture are already well understood, so very little risk is involved in achieving the desired result. Furthermore, the carefully thought plan of developing this tool as a reusable component will guarantee a long lasting value of the tool, both to JPL in particular and to the software engineering field in general.

1.1. Problem Statement

One of the important tasks of software testing is to generate test cases. A test case here refers to a specific set of data used as inputs or environment settings for the system under test (or target system), and can be represented as a set of *parameter-value pairs*. Since a test case parameter may take more than one value, a target system that has N test parameters each has K possible values will require K^N test cases to cover all possible combinations. This exponential growth of test cases makes testing very expensive and highly unmanageable. The challenge here is to generate test cases automatically, and to generate as few of them as possible.

Researchers have employed combinatorial designs to address the problem. One of the best works we found is the commercially available AETG™ system, developed by Telcordia [1][2]. However this tool is an Internet-based interactive service, not in the form of a library, an application program interface (API) or a batch program that one can freely integrate into the desired application.

To remedy this shortcoming, a prototype tool, called TCG (Test Case Generator), is developed by the MDS project by the proposal task lead [3]. The preliminary performance analysis shows that TCG is as powerful as AETG and has a potential to become truly reusable test component. However, this tool has only a prototype available, and further development of it requires additional funding support outside of MDS.

1.2. Justification and Benefits

One of the most important stages in testing of mission software, whether it is for flight or for ground, is the test case generation. The proposed work will save testing work in two respects:

1. The tool will automate test case generation process, which will save testers a lot of work, and the tool will guarantee a complete coverage with selected degree of interaction.
2. The tool will generate as few test cases as possible, this will reduce the number of tests and thus the testing time and resource, sometimes dramatically.

The above have a direct, and very positive, impact on the software testing for almost all projects, especially for the deadline sensitive mission software projects.

This tool is also a fundamentally important link in the testing automation. Many other test automation tools depend on an automatic and effective test case generator like this one.

The tool is not a reinvented wheel because commercial tools are not designed as a reusable component that can be integrated into a test tool set. For example, AETG is implemented as an Internet-based interactive service, not a reusable program.

Furthermore, the use of standard language constructs, such as the standard constraint language OCL (Object Constraint Language) which is path of the standard UML (Unified Modeling Language), allows our tool to last a long time, unlike AETG which uses its proprietary constraint language.

1.3. Estimate of Return on Investment

We believe this work will greatly benefit JPL mission software testing with minimum investment. The most risky phase of the TCG work is now complete. The proposed phase, which is the most crucial phase, of the work will thus bear little or no risk and yet the product is known to be useful.

The estimated cost of carrying out the work is six man-months, the cost is estimated to be on the order of $50-60K.

To quantify the benefit, we compare the use of AETG and the use of TCG – though they are not completely an apple-to-apple comparison. A single seat AETG license with needed “advanced feature” is about $3.5K per year. Assume MDS, X2000 and OPSP will share 10 seats, this figure comes to $35K per year. So the investment on TCG development will break even with return in about two years. However, note that AETG does not fulfill all our need and thus the investment return period for TCG should be adjusted shorter than this.

2. Work Plan and Technical Approach

The proposed task is to implement the already concept proved TCG algorithm, extend it with features including "seed cases," n-way interaction and constraint capabilities, and build a robust tool using Java language (same as the prototyped TCG tool).

To summarize the work steps involved, let's list the following items:

- User and tool interface – including parsing and permuting of the input data and file selection.
- Complete algorithm development – this includes seeds, n-way interaction, and constraints capabilities
- Implement the algorithm in Java with all features in the algorithm
- Add graphical user interface
- Document the work – this include a user's guide and a reference guide

Details are described below.

2.1. Background

A test case can be represented as a set of *parameter-value pairs*. The term "test parameter" is the same as a "category" as in [4]; and a "value" is similar to the term "choice" in [4] with the following difference: a choice normally refers to a range of data while a value is always a single value selected to represent that range. For example, if a parameter or category *X* has a choice "*X*>100", we may select a single value "*X*=101" to represent such a choice in a test case. As an example of the test parameter value pairs (or table), see Table 1 below:

PARAMETER	V_1	V_2	V_3	V_4	V_5
P1: Slew_duration	30	120	300	600	1200
P2: Rax_start	ST1 (37063140)	ST2 (42333540)	ST3 (63068400)		
P3: SPE	1500	2000	2500		
P4: Nav_win_duration	2	5	7		
P5: Nav_win_start	1	2	4		
P6: Exp_per_LOS	3	4	5		
P7: Exp_duration	T1 (9PT83S)	T2 (1PT86M)	T3 (55PT9S)		
P8: Random_seed	R1 (1621913546)	R2 (1944411321)	R3 (1783815836)		
P9: Exec_latency	1	5	10		
P10: Micas_mode	O (OFF)	R (READY)			
P11: Micas_avail	F (FALSE)	T (TRUE)			

Table 1. The parameter-value table for the TCG example

A test case algorithm is designed, based on combinatorial design approach. A prototype is implemented for the MDS project, called MDS TCG. For the above example, the TCG prototype tool generates 20 test cases as shown in Table 2 below:

TEST CASE	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*	*P11*
1	30	ST1	1500	2	1	3	T1	R1	1	O	F
2	120	ST2	2000	5	2	4	T2	R2	5	R	T
3	300	ST1	2000	7	4	5	T3	R3	10	O	T
4	600	ST3	1500	5	4	3	T2	R3	1	R	F
5	1200	ST1	2500	5	1	4	T3	R1	5	R	F
6	120	ST3	2500	2	4	5	T1	R2	10	O	T
7	600	ST2	2500	7	1	3	T2	R1	10	O	T
8	1200	ST2	1500	7	2	5	T1	R3	1	R	F
9	300	ST2	1500	2	2	4	T3	R2	1	R	T
10	1200	ST3	2000	2	1	3	T2	R2	5	O	F
11	30	ST2	2500	5	4	4	T1	R3	5	O	T
12	30	ST3	2000	7	2	4	T3	R1	10	R	F
13	300	ST3	2500	5	1	5	T2	R1	5	R	F
14	600	ST1	2000	2	2	4	T1	R2	1	O	T
15	120	ST1	1500	7	1	3	T3	R3	5	O	F
16	1200	ST1	1500	2	4	4	T2	R3	10	R	T
17	300	ST3	2500	7	2	3	T1	R2	1	R	T
18	600	ST2	2000	5	4	5	T3	R1	5	O	F
19	30	ST2	2000	5	4	5	T2	R2	10	O	F
20	120	ST1	1500	2	1	3	T2	R1	1	R	T

Table 2. The 20 test cases generated by the TCG algorithm

Both TCG and AETG use the greedy approach, which simplifies the algorithm and reduces the computation time, yet the result is proven to be close to optimal. The TCG algorithm differs from AETG's mainly in that the former uses a deterministic method while the latter is using random selection. In the AETG algorithm, each partial test case is determined by first generating *M* different candidate test cases and then choosing one that covers the most new pairs, where *M* is selected to be 50 for best result. The number of candidates TCG needs to generate is equal to the largest cardinality of the parameters, or 5 to 7 in our four examples. This is about one order of magnitude lower than the 50 required by the AETG algorithm.

Several examples were used to test out the TCG algorithm, these were taken from the 6-day and the 12-hour plans generated by an automatic planner used in the Deep Space One (DS1) mission's Remote Agent Experiment (RAX) [6]. We run these examples through both the TCG and the AETG tools. We also collected the test cases generated by the DS1 RAX team for the first three examples so we can make comparison. See Table 3 below for the result.

	TCG	**AETG**	**RAX**
RAX 12-hour plan	20	19	24
RAX 6-day plan	45	45	60
RAX 6-day back-to-back plan	30	30	41
RAX 6-day – short	33	34	N/A

Table 3. Performance comparison using the four examples

From Table 3, we observed that TCG and AETG have comparable performance, and both TCG and AETG are better than the RAX home brewed test case generator by a margin around 25%.

2.2. New features to be implemented by the proposed work

While the prototype tool does the core job, it still needs several important features to become practically useful. Among these features, the following three are of the highest priority:

1. *Seeds*. A seed is a pre-determined test case, either selected by hand or generated by an algorithm. The seeds are useful when the human tester select them deliberately for at least two reasons. The first reason is to assign different priorities to the test inputs. The second reason is to use the same procedure to cope with a dynamic target system. When the target system is updated with extra values for a certain parameter, the old test cases can be used as seeds.

2. *N-way coverage*. This means all value combinations of any *n* parameters should be included in some test case. We had already dealt with the 2-way coverage in the form of (V_x, V_y), and the goal is to extend it to the *n*-way coverage, or (V_{x1}, V_{x2}, … V_{xn}). This *n* is larger than or equal to 1. When *n*=1 the solution is trivial in that you only need as many as *M* test cases where *M* is the largest cardinality of the parameters.

3. *Constraints*. A constraint defines the boundary between allowed and disallowed parameter-value combinations. To implement this kind of constraint, the tool has to be able to partition the parameter-value pairs for the related parameters according to the constraints. Then we need a step in the algorithm to check that the parameter-value selection will not violate such partitions when constructing relevant parameters.

2.3. Detailed Work Plan

Based on the above description, the proposed work will need to include the following items:

1. User and tool interface:

The TCG prototype program was implemented as a Java program. It works if inputs are permuted and entered as data structures by hand. This is fine for experiment but not for production software. To make it useful to other people, the interface has to be built so that it would parse user's test specification file correctly, will permute input parameter-value table, and will allow user to set options and to specify a file (or files) from the file hierarchy.

2. Complete algorithm development:

The prototyping TCG algorithm covers only pair-wise interaction and cannot take seeds or constraints, as said above. To add seeds is not hard, it needs us to mark covered pairs in our data structures before generating any new test case. To add n-way interaction is a bit trickier, the whole data structures designed in the prototype are fixed and cannot accommodate an arbitrary "n." So we need a major data structure rework for this purpose. Finally the constraints part, we first need to define what OCL [5] expressions we allow and define a parser to parse the constraint input, then we need a way to mark the parameter-value pairs with the "impossible" mark, and rework our algorithm to avoid such mark when constructing test cases. This is the main part of the proposed task.

3. Program restructuring to accommodate the above-mentioned new features of the algorithm, implement the code in Java code, which is the same as the language we used for the prototype TCG.

4. Add graphical user interface to the program so user could easily use it or interface with it.

5. Document the work – this includes a user's guide and a reference guide, and a final report. We may provide demos and training to users as appropriate.

2.4. Development Phase Beyond the Proposed Task:

MDS and other JPL projects can readily use the tool at the completion of this proposed work. However, since the tool is designed as a reusable component, it can be further developed with other related test tools to form a powerful testing environment. The main idea is that a set of carefully selected or defined test tools will work synergistically to reach a higher level of power – more powerful than the sum of each component.

Consider the components in the Figure 1 below:

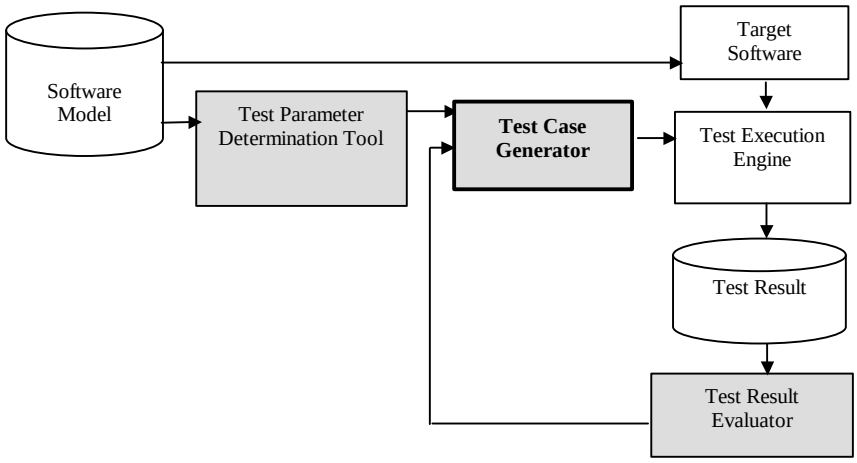

Figure 1. Test Case Generator used as a component

Here the "Test parameter determination tool" will automatically extract information from model, and will select important state variables as "parameters" and build their possible values with them as the parameter-value table. Constraints will be inferred from the model as well. This parameter-value table and constraints will form the input to our TCG tool.

The "Test Result Evaluator," or called "Test Result Analyzer," will take the results of the test case execution. In case one or more failures were found, this tool will determine which test parameter or parameters are at fault. This tool may further analyze the source of the faults, if the code is properly instrumented.

2.5. Partnership and Participation by Others

The ideas of the two related testing tools mentioned above, the test parameter determination tool and the test result evaluator (or sometimes under different names), are currently being pursuit by other people within JPL, e.g. Martin Feather, Allen Nikora, and Ben Smith, to name a few. Their works are closely related to ours and communication with these people has already been well established. We are expecting a highly automated testing environment to be built within JPL, and this proposed work appears to be one important fundamental step.

3. Statement of Endorsement

The following statement is from MDS verification team lead, Wafa Aldiwan, who is our primary customer:

From: Wafa Aldiwan, MDS Verification Lead

To: CSMISS Review Board

Re: Automated Test Case Generator

The TCG tool addresses a critical need for testing and validating all MDS software, at both unit level and system level. We intend to use this tool as soon as it becomes available, and I foresee that the tool will be widely used within JPL – MDS, X2000, OPSP and many more projects to come.

Sincerely,

Wafa Aldiwan

Verification Lead, MDS

4. Schedule and Deliverables

The schedule and deliverables are found in the following schedule diagram.

<Month in the year 2000>	**Milestones**	**Deliverables**
March	User and Tool Interface built; Algorithm enhancement started; The part-time programmer recruited.	
April	Seeds capability completed and tested;	
May	Define constraints using OCL; enhance the algorithm with constraints;	
June	Constraints capabilities completed and tested; Current version will be delivered to MDS;	A version with command line UI, pair-wise interaction, seeds and constraints
July	n-way capability completed	
August	n-way capability tested; graphical user interface added;	
September	Complete product with documents will be delivered to MDS, X2000, OPSP and other customers. Demos and Training will be provided.	Complete TCG tool with GUI and command line UI, n-way interaction, seeds and constraints; also complete document includes user's guide and reference guide

5. Cost Proposal

This task needs about six (6) man-months to be split between the task lead and a programmer to be recruited. Total cost is about $50-60K

5.1. Personnel

The task lead will charge 30-50% of his time on the task. A part-time programmer may charge 50-70% on his/her time. It is assumed that the recruiting process will take about one month. But in the case that it is impossible to find a qualified candidate soon enough, the task lead can charge up to 70% of his time to do this work. In the worst case when de-scope becomes necessary, the n-way interaction and the graphical user interface may be postponed to the next phase of development, and the tool will still be useful.

The task lead job (Yu-Wen Tung):

- Plan, monitor and manage the task; make sure the product is delivered on schedule;
- Enhance the TCG algorithm and design the software architecture and constraint OCL constructs for the programmer to implement;
- Write user's guide and reference guide, hold demos and provide training to the customers

The programmer's job:

- Implement user and tool interface, seeds, constraints and n-way interaction capabilities, and graphical user interface tool as directed by the task lead
- Thoroughly test the tool, deliver the tool to customers and help to perform demos and training.

5.2. Facility, Equipment and Other Charges

No special facility or equipment is needed to perform the task, and no travel is foreseen. However, we assume that the regular monthly charges for PCs and/or workstations (e.g. OAODNS fees), AFS computer accounts and software license fees, book purchasing and things of this sort will be charged to the fund in proportion to the time card charges.

6. Sources of Other Co-funding and Follow-on Funding

It is understood that if the proposed task is funded, MDS will provide support to the task lead as well as the to-be-identified programmer for their remaining time, to work on this or other similar tasks as appropriate. Follow-on funding is also possible from MDS if the proposed tool becomes a reality. That is, when we complete the tool implementation by September as proposed.

7. References

[1] D. M. Cohen et. al., "The AETG System: An Approach to Testing Based on Combinatorial Design," *IEEE Transactions on Software Engineering*, 23(7):437-444, July 1997.

[2] D. M. Cohen et. al., "The Combinatorial Design Approach to Automatic Test Generation," *IEEE Software*, Vol. 13, No. 5, pp. 83-87, September 1996.

[3] Y. Tung and W. Aldiwan, "Automating Test Case Generation for the New Generation of Mission Software," in *Proceedings of the 2000 IEEE Aerospace Conference*, March 2000.

[4] T. J. Ostrand and M. J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Tests," *Communications of the ACM*, 31(8):676-686, June, 1988.

[5] J. Warmer and A. Kleppe, "The Object Constraint Language – Precise Modeling with UML," Addison Wesley Longman, Inc., Reading, Massachusetts, 1999.

[6] B. Smith et. al., "Validation and Verification of the Remote Agent for Spacecraft Autonomy," in *Proceedings of the 1999 IEEE Aerospace Conference*, 1999.